RCKMPI User Manual

Isaías A. Comprés Ureña, Intel Braunschweig
28 of January, 2011

1. Introduction

RCKMPI is a modified MPICH2 for use on the Single Chip Cloud (SCC) computer from Intel. It takes into account the memory organization and the presence the message passing (MPB) buffer on the SCC, to allow for low latency and high bandwidth MPI traffic.

This document introduces the three new SCC specific channels provided. It also describes the necessary steps to configure, compile and install the library, as well as to compile and run MPI applications.

2. RCKMPI Channels

In this section, the channel specific characteristics are described. There are 4 channels available: SOCK, SCCMPB, SCCSHM and SCCMULTI. The SOCK channel is kept from the MPICH2 tree; it uses the TCP/IP stack for communication. The rest are custom for the SCC, using the MPB, shared memory, or a combination of both. Multiple channel operation is done only in the SCCMULTI channel and is based on message size.

POPSHM is used in all shared memory based communications and is documented separately. Because of this, the use of the SCCSHM and SCCMULTI channels require a special Linux kernel that enables the use of POPSHM pages. Note that this new kernel is not yet publicly available.

SCCMPB

This channel uses the message passing buffer for both signaling and payload. It depends only on the SCC low level API, and assumes exclusive ownership of the message passing buffer, meaning that no other applications are allowed to use the MPB when running MPI jobs with this channel.

The SCCMPB is the default channel, for a couple of reasons:

- It does not require a POPSHM patched kernel.
- Performance is the best for small messages.
- It is the most thoroughly tested channel.
- It requires the least memory to operate.

The SCCMPB channel has shown the best overall performance in practice. MPI applications typically use smaller than 4KB messages and these are always faster on the MPB channel, in comparison with the others available. If the target application uses larger messages frequently, then the SCCMULTI channel is recommended.

SCCSHM

This channel is based on purely shared memory communication. The kind of shared memory used, both in SCCSHM and SCCMULTI, is called POPSHM. This is a new way of sharing memory that was

developed in parallel with the RCKMPI library. It is documented separately and can be used independently of RCKMPI.

SCCSHM is not a well performing channel; it is only faster than the SOCK channel. It can outperform the SCCMPB in specific situations, very rarely in practice. It is never faster than the SCCMULTI channel. For these reasons, this channel is only recommended for understanding POPSHM performance, and for comparison purposes.

SCCMULTI

This channel truly exploits the peculiarities of the SCC's memory organization. It uses the MPB for control flow and payload, and shared memory for payload. As with the SCCSHM channel, the shared memory used is based on the POPSHM library and requires a patched kernel.

Shared memory or MPB buffers are used, depending of the message size, for payload. In that sense, it has multichannel operation. Multiple channel operation differs from other MPICH2 compatible channels in that, the decision is made based on message size, not on the location of the process.

This channel is a good choice when larger messages are used by an application. For smaller messages, it is slightly slower than the MPB only channel. It has been observed to be considerable faster for large messages, especially for increased process counts.

3. Setup

Setting up RCKMPI on the SCC can be done in 3 steps: configuration, compilation, and loading.

After all binaries are loaded in the SCC, it is recommended to start an MPD process ring, so that MPI jobs can be launched quickly. The creation of the ring, and the use of the MPD are completely optional, but highly recommended, since all internal testing has been done in this way.

Other methods besdes an MPD process ring include the follwiong.

- Hydra (requires Python), which is used by default in the newer releases of MPICH2, that is, 1.3x . However, there are some issues with Hydra, since it depends on global variables for most of its configuration, and the Dropbear SSH client found in the default Linux image, does not handle the environment well.
- SMPD. This is a C based process manager. The advantage is that you don't need Python, but it's the oldest PM and does not have the features of the MPD or Hydra.

Prerequisites

To use the multi-purpose daemon (MPD), Python is required. Python and its dependencies can be obtained in the MARC community site, the file is called: `prereqs.tar.gz` .

While using the MCPC, extract its contents somewhere in `/shared` (referred as `<extracted directory>` below). Inside the SCC and at each core (this can be achieved by either broadcasting the commands using sccKonsole, or with PSSH), copy (with a `cp -a`) the following files into `/usr/bin/`:

```
/shared/<extracted directory>/python/bin/*
```

Also inside the SCC, copy the following files into `/usr/lib/`:

```
/shared/<extracted directory>/python/lib/*
/shared/<extracted directory>/libssl/lib/*
/shared/<extracted directory>/zlib/lib/*
```

Copy MPD configuration file to `/etc/` (the file can be customized for your needs):

```
conf/mpich/mpd.conf
```

Note that, using SMPD or HYDRA process managers is not covered in this manual. If using SMPD, Python is not required at all.

These steps may be unnecessary in the future. Work is ongoing to change the current SCC Linux setup to include Python, and be flexible enough to add other tools as well, including the RCKMPI library.

Configuration

Obtain the RCKMPI library by either downloading a compressed TAR file or through the SVN repository. After extraction or check out, the configuration script can be run. The library sources can be located anywhere in the MCPC.

Before you run the configure script, you need to have a working cross compiling environment. It is highly recommended that you test your environment before running the configure scripts, since this can take a while, and most errors will only show up when compiling or running target applications, not when compiling the library itself.

To configure the library, move to the `rckmpi` folder and run the `configure` script. The configure script is nearly identical to that of MPICH2. The notable difference is the availability of the new channels. To see the available options, run:

```
./configure --help
```

Quick Solution

To use the default SCCMPB channel, and the MPD process manager, just modify the following command to have the desired install directory:

```
./configure --prefix=/shared/<install path>/rckmpi --with-pm=mpd
```

Install Directory

The install directory can be passed with the prefix option. The install directory should be in `/shared` so that it can be seen by both the MCPC and the SCC system. The following can be used as a template:

```
./configure --prefix=/shared/<install path>/rckmpi
```

Channel Selection

If no channel is specified, like in the previous example, then the SCCMPB channel is used. To use one of the other channels (given that you are running a POPSHM capable kernel), `--with-device` can be used as follows:

```
./configure --prefix=/shared/<install path>/rckmpi \
      --with-device=ch3:<channel name>
```

Where `<channel name>` can be `sock`, `sccmpb`, `sccshm` or `sccmulti`. Specifying `sccmpb` is the same as not using the `-with-device` flag.

Even without POPSHM, you can specify –with-device=ch3:sock.

Process Manager Selection

By default, all process managers are compiled and added to the library. Typically, only the MPD is used, since it has some advantages when working on the SCC:

- The setup is done only once, after loading the binaries.
- No host file is necessary after ring is set up.
- Job startup is fast after ring set up.
- Issues with the Dropbear SSH client (found in the default SCC Linux image) and environment variables are avoided.

The process manager can be specified in `configure`, and only the one specified will be built into the library, saving memory. This can be done with the `-with-pm` option as follows:

```
./configure --prefix=/shared/<install path>/rckmpi --with-pm=mpd
```

Advanced Channel Options

There are extra features that can be enabled in the channels (they are common to all 3 new channels, have no effect on the `sock` channel). These are: relaxed polling, statistics, print statistics and watchdog timer. These can be enabled by defining the following global variables, with the use of the CFLAGS variable, before running `configure`:

- `WATCHDOG`: Enables the watch dog counter. This counter is used to detect deadlocks.
- `RELAXED_POLLING`: Enables the relaxed polling strategy, without statistics.
  - `STATISTICS`: Calculates statistics and uses the gathered information to modify polling relaxation. Only works if `RELAXED_POLLING` is enabled.
    - `PRINT_STATISTICS`: Generates a file with the statistics information after an MPI job is completed. Only works if `STATISTICS` and `RELAXED_POLLING` are enabled.

For example, to enable relaxed polling, include a definition of `RELAXED_POLLING` in the CFLAGS variable, with the use of the `-D` compiler directive:

```
export CFLAGS="-march=pentium -DRELAXED_POLLING"
```

Similarly, when using CSH:

```
setenv CFLAGS "-march=pentium -DRELAXED_POLLING"
```

Watch Dog Counter

This feature will add a watch dog counter and related logic to detect deadlocks. It can detect deadlocks in any MPI application, so it can be useful to MPI developers.

Relaxed Polling

Polling operations are done hundreds of times, depending on the MPI application, before a message is received. This feature relaxes polling frequency, independently per remote process, depending on some criteria.

If this feature is enabled alone, the relaxation is based on the time the last message was received, and if a multiple message packet is in transit.

Statistics

If relaxed polling is enabled, the statistics strategy can be also specified. In this case, the channel samples the polling successes (a message was available for reception after a poll) and calculates the average probability that a message will be received after a poll, for each remote process.

Based on this information, the polling frequency is adjusted for each remote process.

Print Statistics

If both relaxed poling and the statistics strategy are selected, it can also be specified to generate a file containing the internal sampled data, calculated probabilities and polling adjustments done by the library.

One file is generated per process of an MPI job. This can be useful for observing patterns in higher level implementations of collective operations and typical MPI traffic behavior for different applications.

Compilation

To compile the library, in the `rckmpi` directory, run the make script:

```
make && make install
```

This will build the library, and later install it to the directory specified by the `--prefix` flag in the configure script.

Compiling MPI Applications

To compile MPI applications, the cross compiling environment must be updated with the RCKMPI library and related scripts.

First, update your `PATH` variable. If using BASH, use this template:

```
export PATH=/shared/<install path>/rckmpi/bin/:$PATH
```

For CSH, use the following one:

```
setenv PATH /shared/<install path>/rckmpi/bin/:$PATH
```

Afterwards, updating the `LD_LIBRARY_PATH` variable is required. For BASH:

```
export LD_LIBRARY_PATH=/shared/<install path>/rckmpi/lib/:$LD_LIBRARY_PATH
```

When using CSH:

```
setenv LD_LIBRARY_PATH /shared/<install path>/rckmpi/lib/:$LD_LIBRARY_PATH
```

After this, the `mpicc` (and the FORTRAN related ones as well, depending on your cross compiler environment) should be available.

Loading Binaries to the SCC

After compilation, the library and the compiled binaries must be loaded to the SCC. Note that, if using the MPD process manager (recommended), the prerequisites must also be loaded (refer to previous section "Prerequisites").

By loading, it is meant copying, by means of simple `cp -a` or `rsync`, from `/shared` to the SCC RAM file system. As mentioned earlier, this must be done on each core (either broadcast with sccKonsole or use PSSH, creation of a script for this task is suggested).

Copy into `/usr/bin/`:

```
/shared/<install path>/rckmpi/bin/*
```

Copy into `/usr/lib/`:

```
/shared/<install folder>/rckmpi/lib/*
```

Copy your applications as well (they are typically loaded in the `/root` directory).

Note: Applications can be run directly from the `/shared` directory, at the cost of performance.

4. MPD Ring Setup

To run MPI applications using the MPD, first a ring of daemons must be set up. This can be done in two ways: with or without a host file.

With Host File

Issue the following command (this will take about a minute depending of the process count):

```
mpdboot --totalnum=<process count> --file=<host file> --maxbranch=1
```

The `maxbranch` option will guaranteed that the ring is created in order. This allows for jobs to start with process numbers mapping directly into core numbers (e.g. core 0 to process 0, etc). This may be desirable, but is not required for the proper function of the library.

To verify that the ring is properly set up and to see the order in which they were started, issue a:

```
mpdtrace -l
```

Without Host File

Issue this command in the core that is desired to be the root host:

```
mpd --daemon
```

To verify correct start up and to obtain the current port (necessary later), run:

```
mpdtrace -l
```

Output should be in this format:

```
<root host>_<port> (192.168.0.<machine>)
```

On all nodes (use sccKonsole or PSSH), run:

```
mpd --daemon --host=<root host> --ncpus=1 --port=<root port>
```

Run `mpdtrace` again to verify:

```
mpdtrace -l
```

5. Running Applications

Library compilation and installation needs to be done only once on the MCPC. Loading and MPD ring setup need to be done only once after starting a fresh SCC Linux on the SCC cores.

Only the user applications need to be reloaded if they were changed and recompiled. If performance is not a concern, the user applications can also be run directly from their location on the `/shared` directory.

To run MPI applications, `mpiexec` or `mpirun` can be used normally. For example:

```
mpirun -np 48 ./application
```